

A Framework for Assessing Achievability Of Data-Quality Constraints

Rada Chirkova¹, Jon Doyle¹, and Juan L. Reutter²

¹ Computer Science Department, North Carolina State University
North Carolina, USA

² Pontificia Universidad Católica de Chile
chirkova@csc.ncsu.edu, Jon_Doyle@ncsu.edu, jreutter@ing.puc.cl

Abstract. Assessing and improving the quality of data are fundamental challenges for data-intensive systems that have given rise to numerous applications targeting transformation and cleaning of data. However, while schema design, data cleaning, and data migration are nowadays reasonably well understood in isolation, not much attention has been given to the interplay between the tools addressing issues in these areas. We focus on the problem of determining whether the available data-processing procedures can be used together to bring about the desired quality characteristics of the given data. For an illustration, consider an organization that is introducing new data-analysis tasks. Depending on the tasks, it may be a priority for the organization to determine whether its data can be processed and transformed using the available data-processing tools to satisfy certain properties or quality assurances needed for the success of the task. Here, while the organization may control some of its tools, some other tools may be external or proprietary, with only basic information available on how they process data. The problem is then, how to decide which tools to apply, and in which order, to make the data ready for the new tasks?

Toward addressing this problem, we develop a new framework that abstracts data-processing tools as black-box procedures with only some of the properties exposed, such as the applicability requirements, the parts of the data that the procedure modifies, and the conditions that the data satisfy once the procedure has been applied. We show how common database tasks such as data cleaning and data migration are encapsulated into our framework and, as a proof of concept, we study basic properties of the framework for the case of procedures described by standard relational constraints. We show that, while reasoning in this framework may be computationally infeasible in general, there exist well-behaved special cases with potential practical applications.

1 Introduction

A common approach to ascertaining and improving the quality of data is to develop procedures and workflows for repairing or improving data sets with respect to quality constraints. The community has identified a wide range of

data-management problems that are vital in this respect, leading to the creation of several lines of studies, which have normally been followed by the development of toolboxes of applications that practitioners can use to solve their problems. This has been the case, for example, for the Extract-Transform-Load (ETL) [13,18] process in business applications, or for the development of automatic tools to reason about the completeness or cleanliness of the data [15].

As a result, organizations facing data-improvement problems now have access to a variety of data-management tools to choose from; the tools can be assembled to construct so-called workflows of data operations. However, in contrast with the considerable body of research on particular data operations, or even entire business workflows (see, e.g., [12,8,11,3]), previous research appears to have not focused explicitly either on the assembly process itself or on providing guarantees that the desired data-quality constraints will be satisfied once the assembled workflow of procedures has been applied to the available data.

We investigate the problem of constructing workflows from already available procedures. That is, we consider a scenario in which an organization needs to meet a certain data-quality criterion or goal using available data-improvement procedures. In this case, the problem is to understand whether these procedures can be assembled into a data-improvement workflow in a way that would guarantee that the data set produced by the workflow will effectively meet the desired quality goal.

Motivating example: Suppose that data stored in a medical-data aggregator (such as, e.g., Premier [27]) are accessed to perform a health-outcomes analysis in population health management [19,25,32], focusing on repeat emergency-room visits in the Washington, DC area. The goal of the analysis is to see whether there is a relationship between such repeat visits and ages and zip codes of patients.

We assume that the aggregator imports information about emergency-room visits from a number of facilities, and stores the information using a relation *Visits* with attributes *facility* and *facilityLoc* for the ID and location of the medical facility, *patInsur* for the patient insurance number, and *timestamp* for the date and time of the visit. We also assume that medical-record information imported from each facility is stored at the aggregator in a relation *Patients*, with attributes *facility*, *patInsur*, *name*, *age*, *zipCode*, and so on.

The analyst plans to isolate information about emergency-room visits for the Washington area in a relation *LocVisits*, which would have all the attributes of *Visits* except *facilityLoc*, as the values of the latter are understood to be fixed. Further, to obtain the age and zip code of patients, the analyst also needs to integrate the data in *Visits* with those of *Patients*.

To process the data, the analyst has access to some procedures that are part of the aggregator’s everyday business. For example, the aggregator periodically runs a *StandardizePatientInfo* procedure, which first performs entity resolution on insurance IDs in *Patients*, using both the values of all the patient-related attributes in that relation and a separate “master” relation *InsurerInfo* that stores authoritative patient information from insurance companies, and then merges

the results into *Visits*. Further, the aggregator offers a procedure *MigrateIntoLocVisits* that will directly populate *LocVisits* with the relevant information about emergency rooms (but not the age and zip code of patients).

The analyst is now facing a number of choices, some of which we list here:

- (i) Use the *StandardizePatientInfo* procedure on *Patients*, then manually import the correct(ed) information into *LocVisits*, and finally join this relation with *Patients*.
- (ii) Run *MigrateIntoLocVisits* to get the relevant patient information, and then join with *Patients* without running the procedure *StandardizePatientInfo*.
- (iii) Add *age* and *zipCode* attributes to *LocVisits*, get the information into *LocVisits* as in (ii), and then try to modify *StandardizePatientInfo* into operating directly on *LocVisits*.

Which of these options is the best for the planned analysis? Option (i) seems to be the cleanest, but if the analyst suspects that *StandardizePatientInfo* may produce some loss of data, then going with (ii) or (iii) might be a better option. Further, suppose the analyst also has access to a separate relation *HealthcareInfo* from a health NGO, with information about emergency-room visits gathered from other independent sources. Then the analyst could pose the following quality criterion on the assembled workflow: The result of the workflow should provide at least the information that can be obtained from the relation *HealthcareInfo*. How could one guarantee that such a criterion will be met?

Contributions: Our goal is to develop a general framework that can be used to determine whether the available data-processing tools can be put together into a workflow capable of producing data that meet the desired quality properties. To address this problem, we abstract data-processing tools as black-box procedures that expose only certain properties. The properties of interest include (i) preconditions, which indicate the state of the data required for the procedure to be applicable; (ii) the parts of the data that the procedure modifies; and (iii) postconditions, which the data satisfy once the procedure has been applied.

In this paper we introduce the basic building blocks and basic results for the proposed framework for assessing achievability of data-quality constraints. The contributions include formalizing the notion of (sequences of) data-transforming procedures, and characterizing instances that are outcomes of applying (sequences of) procedures over other instances. We also illustrate our design choices by discussing ways to encode important database tasks in the proposed framework, including data migration, data cleaning, and schema updates.

One of the advantages of our framework is its generality, as it can be used to encode multiple operations not only on relational data, but on semistructured or even unstructured text data. This very generality implies that to be able to reason about the properties of our framework, one needs to first instantiate some of its most abstract components. As a proof of concept, we provide an in-depth analysis of applications of (sequences of) procedures over relational data, where the procedures are stated using standard relational-data formalisms. We show that properties concerning outcomes of procedures are in general (not surprisingly) undecidable. At the same time, we achieve decidability and tractability

for broad classes of realistic procedures that we illustrate with examples. While the formalism and results presented in this paper have practical implications on their own, we see them mainly as prerequisites that need to be understood before one can formalize the notion of assembling procedures in the context of and in response to a user task. We conclude this paper by showing how the proposed framework can be used to formally define the following problem: Given a set of procedures and data-quality criteria, is it possible to assemble a sequence of procedures such that the data outcome is assured to satisfy this criteria?

Related Work: Researchers have been working on eliciting and defining specific dimensions of quality of the data — [31] provides a widely acknowledged standard; please also see [17,24]. At the general level, high-quality data can be regarded as being fit for their intended use [22,30,10] — that is, both context and use (i.e., tasks to be performed) need to be taken into account when evaluating and improving the quality of data. Recent efforts have put an emphasis on information-quality policies and strategies; please see [22] for a groundbreaking set of generic information-quality policies that structure decisions on information. An information-quality improvement cycle, consisting of the define-measure-analyze-improve steps for data quality, has been proposed in [29]. Work has also been done [23] in the direction of integrating process measures with information-quality measures. Our work is different from these lines of research in that in our framework we assume that task-oriented data-quality requirements are already given in the form of constraints that need to be satisfied on the data, and that procedures for improving data quality are also specified and available. Under these assumptions, our goal is to determine whether the procedures can be used to achieve satisfaction of the quality requirements on the data.

The work [15] introduces a unified framework covering formalizations and approaches for a range of problems in data extraction, cleaning, repair, and integration, and also supplies an excellent survey of related work in these areas. More recent work on data cleaning includes [7,6,21,28,26]. The research area of business processes [11] studies the environment in which data are generated and transformed, including processes, users of data, and goals of using the data. In this context, researchers have studied automating composition of services into business processes, see, e.g., [3,4,5], under the assumption that the assembly needs to follow a predefined workflow of executions of actions (services). In contrast, in our work, the predefined part is the specified constraints that the data should satisfy after the assembled workflow of available procedures has been applied to it. Another line of work [12,8] is closer to reasoning about static properties of business process workflows. That work is different from ours in that it does not pursue the goal of constructing new workflows.

Outline of the paper: Section 2 contains basic definitions used in the paper. Section 3 introduces the proposed framework, and Section 4 discusses encoding tasks such as data exchange, data cleaning, and alter-table statements. The formal results are presented in Section 5. Section 6 concludes with a discussion of future challenges and opportunities.

2 Preliminaries

Schemas and instances: Assume a countably infinite set of attribute names $\mathcal{A} = \{A_1, A_2, \dots\}$ and a countably infinite set (disjoint from \mathcal{A}) of relation names $\mathcal{R} = \{R_1, R_2, \dots\}$. A relational schema is a partial function $\mathcal{S} : \mathcal{R} \rightarrow 2^{\mathcal{A}}$ with finite domain, which associates a finite set of attributes to a finite set of relation symbols. If $\mathcal{S}(R)$ is defined, we say that R is in \mathcal{S} . A schema \mathcal{S}' extends a schema \mathcal{S} if for each relation R such that $\mathcal{S}(R)$ is defined, we have that $\mathcal{S}(R) \subseteq \mathcal{S}'(R)$. That is, \mathcal{S}' extends \mathcal{S} if \mathcal{S}' assigns at least the same attributes to all relations in \mathcal{S} . We also assume a total order $\leq_{\mathcal{A}}$ over all attribute names in order to be able to switch between the *named* and *unnamed* perspectives for instances and queries.

We define instances so that it is possible to switch between the named and unnamed perspectives. Assume a countably infinite set of domain values D (disjoint from both \mathcal{A} and \mathcal{R}). Following [1], an instance I of schema \mathcal{S} assigns to each relation R in \mathcal{S} , where $\mathcal{S}(R) = \{A_1, \dots, A_n\}$, a set R^I of *named* tuples, each of which is a function of the form $t : \{A_1, \dots, A_n\} \rightarrow D$, representing the tuples in R . (We use $t(A_i)$ to denote the element of a tuple t corresponding to the attribute A_i .) By using the order $<_{\mathcal{A}}$ over attributes, we can alternatively view t as an *unnamed* tuple, corresponding to the sequence $\bar{t} = t(A_1), \dots, t(A_n)$, with $A_1 <_{\mathcal{A}} \dots <_{\mathcal{A}} A_n$. Thus, we can also view an instance I as an assignment R^I of sets of unnamed tuples (or just tuples) $\bar{t} \in D^n$. In general, when we know all attribute names for a relation, we use the unnamed perspective, but when the set of attributes is not clear, we resort to the named perspective. For the sake of readability, we abuse notation and use $\text{Schema}(I)$ to denote the schema of an instance I .

For instances I and J over a schema \mathcal{S} , we write $I \subseteq J$ if for each relation symbol R in \mathcal{S} we have that $R^I \subseteq R^J$. Furthermore, if I_1 and I_2 are instances over respective schemas \mathcal{S}_1 and \mathcal{S}_2 , we denote by $I_1 \cup I_2$ the instance over schema $\mathcal{S}_1 \cup \mathcal{S}_2$ such that $R^{I_1 \cup I_2} = R^{I_1} \cup R^{I_2}$ if R is in both \mathcal{S}_1 and \mathcal{S}_2 , $R^{I_1 \cup I_2} = R^{I_1}$ if R is only in \mathcal{S}_1 , and $R^{I_1 \cup I_2} = R^{I_2}$ if R is only in \mathcal{S}_2 . Finally, an instance I' *extends* an instance I if (1) $\text{Schema}(I')$ extends $\text{Schema}(I)$, and (2) for each relation R in $\text{Schema}(I)$ with assigned attributes $\{A_1, \dots, A_n\}$ and for each tuple t in R^I , there is a tuple t' in $R^{I'}$ such that $t(A_i) = t'(A_i)$ for each $1 \leq i \leq n$. Intuitively, I extends I' if the projection of I' over the schema of I is contained in I .

Conjunctive queries: Since our goal is for queries to be applicable to different schemas, we adopt a named perspective on queries. A *named atom* is an expression of the form $R(A_1 : x_1, \dots, A_k : x_k)$, where R is a relation name, each A_i is an attribute name, and each x_i is a variable. We say that the variables mentioned by such an atom are x_1, \dots, x_k , and the attributes mentioned by it are A_1, \dots, A_k . A conjunctive query (CQ) is an expression of the form $\exists \bar{z} \phi(\bar{z}, \bar{y})$, where \bar{z} and \bar{y} are tuples of variables and $\phi(\bar{z}, \bar{y})$ is a conjunction of named atoms that use the variables in \bar{z} and \bar{y} .

A named atom $R(A_1 : x_1, \dots, A_k : x_k)$ is *compatible* with schema \mathcal{S} if $\{A_1, \dots, A_k\} \subseteq \mathcal{S}(R)$. A CQ is compatible with \mathcal{S} if all its named atoms are compatible. Given a named atom $R(A_1 : x_1, \dots, A_k : x_k)$, an instance I of a schema

\mathcal{S} that is compatible with the atom, and an assignment $\tau : \{x_1, \dots, x_k\} \rightarrow D$ of values to variables, we say that (I, τ) satisfy $R(A_1 : x_1, \dots, A_k : x_k)$ if there is a tuple $a : \mathcal{A} \rightarrow D$ matching values with τ on attributes in R in the sense that $a(A_i) = \tau(x_i)$ for each $1 \leq i \leq k$. (Under the unnamed perspective we would require a tuple a in R^I such that its projection $\pi_{A_1, \dots, A_k} a$ over attributes A_1, \dots, A_k is precisely the tuple $\tau(x_1), \dots, \tau(x_k)$.) The usual semantics of conjunctive queries now follows, extending the notion of assignments in the usual way. Finally, given a conjunctive query Q that is compatible with \mathcal{S} , the evaluation $Q(I)$ of Q over I is the set of all the tuples $\tau(x_1), \dots, \tau(x_k)$ such that (I, τ) satisfy Q .

We also need to specify queries that extract all tuples stored in a given relation, regardless of the schema, as is done in SQL with the query `SELECT * FROM R`. To be able to do this, we also use what we call *total queries*, which, as we do not need to know the arity of R , are simply constructs of the form R , for a relation name R . A total query of this form is compatible with a schema \mathcal{S} if $\mathcal{S}(R)$ is defined, and the evaluation of this query over an instance I over a compatible schema \mathcal{S} is simply the set of tuples R^I .

Data Constraints: Most of our data constraints can be captured by *tuple-generating dependencies (tgds)*, which are expressions of the form $\forall \bar{x} (\exists \bar{y} \phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x}, \bar{z}))$, for conjunctive queries $\exists \bar{y} \phi(\bar{x}, \bar{y})$ and $\exists \bar{y} \psi(\bar{x}, \bar{z})$, and by *equality-generating dependencies (egds)*, which are expressions of the form $\forall \bar{x} (\exists \bar{y} \phi(\bar{x}, \bar{y}) \rightarrow x = x')$, for a conjunctive query $\exists \bar{y} \phi(\bar{x}, \bar{y})$ and variables x, x' in \bar{x} . As usual, for readability we sometimes omit the universal quantifiers of tgds and egds. An instance I satisfies a set Σ of tgds and egds, written $I \models \Sigma$, if (1) the schema of I is compatible with each conjunctive query in each dependency in Σ , and (2) every assignment $\tau : \bar{x} \cup \bar{y} \rightarrow D$ such that $(I, \tau) \models \phi(\bar{x}, \bar{y}) \rightarrow D$ can be extended into an assignment $\tau' : \bar{x} \cup \bar{y} \cup \bar{z} \rightarrow D$ such that $(I, \tau') \models \psi(\bar{x}, \bar{z})$.

A tgd is *full* if it does not use existentially quantified variables on the right-hand side. A set Σ of tgds is *full* if each tgd in Σ is full. Σ is *acyclic* if an acyclic graph is formed by representing each relation mentioned in a tgd in Σ as a node and by adding an edge from node R to S if a tgd in Σ mentions R on the left-hand side and S on the right-hand side.

Structure Constraints: Structure constraints are used to specify that schemas need to contain a certain relation or certain attributes. A structure constraint is a formula of the form $R[\bar{s}]$ or $R[*]$, where R is a relation symbol, \bar{s} is a tuple of attributes, and $*$ is a symbol not in \mathcal{A} or \mathcal{R} intended to function as a wildcard. A schema \mathcal{S} satisfies a structure constraint $R[\bar{s}]$, denoted by $\mathcal{S} \models R[\bar{s}]$, if $\mathcal{S}(R)$ is defined, and each attribute in \bar{s} belongs to $\mathcal{S}(R)$. The schema satisfies the constraint $R[*]$ if $\mathcal{S}(R)$ is defined. For an instance I over a schema \mathcal{S} and a set Σ of tgds, egds, and structure constraints, we write $(I, \mathcal{S}) \models \Sigma$ if I satisfies each data constraint in Σ and \mathcal{S} satisfies each structure constraint in Σ .

3 Procedures

In this section we formalize the notion of procedures that transform data. We view procedures as black boxes, and assume no knowledge of or control over their inner workings. Our reasoning about procedures is based on two properties: an input condition, or *precondition* on the state of the data that must hold for a procedure to be applicable, and an output condition, or *postcondition* on the state of the data that must hold after the application.

Example 1. Consider again the medical example discussed in the introduction, with a schema having two relations: *LocVisits*, holding information about emergency-room visits in a geographical area, and *EVisits*, holding visit information for an individual emergency room in a particular location. Suppose we know that a procedure is available that migrates the data from *EVisits* to *LocVisits*. We do not know how the procedure works, but we do know that once it has been applied, all tuples in *EVisits* also appear in *LocVisits*. In other words, this procedure can be described by the following information:

Precondition: The schema has relations *LocVisits* and *EVisits*, each with attributes *facility*, *patInsur* and *timestp* (standing for facility ID, patient insurance ID, and timestamp).

Postcondition: Every tuple from *EVisits* is in *LocVisits*.

Scope and safety guarantees: To rule out procedures that, for example, delete all the tuples from the database, we must be assured that our procedure only modifies the relation *LocVisits*, and that it preserves all the tuples present in *LocVisits* before the application of the procedure. We shall soon see how to encode these guarantees into our framework.

Suppose that after a while, the requirements of one of the partner agencies of the organization impose an additional requirement: Relation *LocVisits* should also contain information about the age of the patients. Suppose the organization also has a relation *Patients*, where the patient age is recorded in attribute *age*, together with *facility* and *patientId*. To migrate the patient ages into *LocVisits*, one needs the following steps: First add the attribute *age* to *LocVisits*, and then update this table so that the patient ages are as recorded in *Patients*. We observe that all the procedures involved in this operation can be captured using the same framework of preconditions, postconditions, and scope/safety guarantees that we used to capture the data-migration procedure.

3.1 Formal Definition

We define procedures with respect to a class \mathbb{C} of constraints and a class \mathbb{Q} of queries.

Definition 1. A procedure P over \mathbb{C} and \mathbb{Q} is a tuple $(Scope, \mathcal{C}_{in}, \mathcal{C}_{out}, \mathcal{Q}_{safe})$, where

- *Scope* is a set of structure constraints that defines the scope (i.e., relations and attributes) in which the procedure acts;

- \mathcal{C}_{in} and \mathcal{C}_{out} are constraints in \mathbb{C} that describe the pre- and postconditions of the procedure, respectively; and
- \mathcal{Q}_{safe} is a set of queries in \mathbb{Q} that serve as a safety guarantee for the procedure.

Example 2. Let us return to the procedure outlined in Example 1, where the intention was to define migration of data from relation *EVisits* into *LocVisits*. In our formalism, we describe this procedure as follows.

Scope: Since the procedure migrates tuples into *LocVisits*, the scope of the procedure is just this relation. This is described using the structure constraint *LocVisits*[*].

\mathcal{C}_{in} : We use the structure constraints *EVisits*[*facility, patInsur, timestp*] and *LocVisits*[*facility, patInsur, timestp*], to ensure that the database has the correct attributes.

\mathcal{C}_{out} : The postcondition comprises the *tg*d

$$EVisits(facility : x, patInsur : y, timestp : z) \rightarrow LocVisits(facility : x, patInsur : y, timestp : z).$$

That is to say, after the procedure has been applied, the projection of *EVisits* over *facility*, *patInsur* and *timestp* is a subset of the respective projection of *LocVisits*.

\mathcal{Q}_{safe} : We can add safety guarantees in terms of queries that need to be preserved when the procedure is applied. In this case, since we do not want the procedure to delete anything that was stored in *LocVisits* before the migration, we add the safety constraint *LocVisits*(*facility* : *x*, *patInsur* : *y*, *timestp* : *z*), whose intent is to state that all answers to this query on *LocVisits* that are present in the database before the application of the procedure must be preserved. We formalize this intuition when giving the semantics below.

3.2 Semantics

Formalizing the semantics of procedures requires additional notation. Given a set \mathcal{C} of structure constraints and a schema \mathcal{S} , we denote by $Q_{\mathcal{S} \setminus \mathcal{C}}$ the conjunctive query that, intuitively, is meant to retrieve the projection of the entire database over all relations and attributes not mentioned in \mathcal{C} . Formally, $Q_{\mathcal{S} \setminus \mathcal{C}}$ includes a conjunct $R(A_1 : z_1, \dots, A_m : z_m)$ for each relation R in \mathcal{S} but not mentioned in \mathcal{C} , where $\mathcal{S}(R) = \{A_1, \dots, A_m\}$ and z_1, \dots, z_m are fresh variables. In addition, if some constraint in \mathcal{C} mentions a relation T in \mathcal{S} , but no constraint in \mathcal{C} is of the form $T[*]$, then $Q_{\mathcal{S} \setminus \mathcal{C}}$ also includes a conjunct $T(B_1 : z_1, \dots, B_k : z_k)$, where $\{B_1, \dots, B_k\}$ is the set of all the attributes in $\mathcal{S}(T)$ that are not mentioned in any constraint in \mathcal{C} , and z_1, \dots, z_k are again fresh variables. For example, consider a schema \mathcal{S} with relations R , S , and T , where R has attributes A_1 and A_2 , T has attributes B_1 , B_2 and B_3 , and S has A_1 and B_1 . Further, consider the set \mathcal{C} with a single constraint $R[*] \wedge S[B_1]$. Then $Q_{\mathcal{S} \setminus \mathcal{C}}$ is the query $T(B_1 : z_1, B_2 : z_2, B_3 : z_3) \wedge S(A_1 : w_1)$. Note that $Q_{\mathcal{S} \setminus \mathcal{C}}$ is unique up to the renaming of variables and order of conjuncts.

<i>EVisits</i>			<i>LocVisits</i>		
<i>facility</i>	<i>patInsur</i>	<i>timestp</i>	<i>facility</i>	<i>patInsur</i>	<i>timestp</i>
1234	33	070916 12:00	1234	33	070916 12:00
2087	91	090916 03:10	1222	33	020715 07:50

(a) Instance I

<i>EVisits</i>			<i>LocVisits</i>		
<i>facility</i>	<i>patInsur</i>	<i>timestp</i>	<i>facility</i>	<i>patInsur</i>	<i>timestp</i>
1234	33	070916 12:00	1234	33	070916 12:00
2087	91	090916 03:10	1222	33	020715 07:50
			2087	91	090916 03:10

(b) Possible outcome J_1 of applying P over I

<i>LocVisits</i>		
<i>facility</i>	<i>patInsur</i>	<i>timestp</i>
1234	33	070916 12:00
1222	33	020715 07:50
2087	91	090916 03:10
4561	54	080916 23:45

(c) relation *LocVisits* in J_2

<i>LocVisits</i>			
<i>facility</i>	<i>patInsur</i>	<i>timestp</i>	<i>age</i>
1234	33	070916 12:00	21
1222	33	020715 07:50	45
2087	91	090916 03:10	82

(d) relation *LocVisits* in J_3

Fig. 1. Instance I of Example 3 (a), a complete possible outcome (b), and the relation *LocVisits* of two other possible outcomes, one in which *LocVisits* contains additional tuples not mentioned in *EVisits* (c), and one where an extra attribute is added to *LocVisits* (d).

A procedure $P = (\text{Scope}, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{safe}})$ is *applicable* on an instance I over schema \mathcal{S} if (1) The query $Q_{\mathcal{S} \setminus \text{Scope}}$ and each query in $\mathcal{Q}_{\text{safe}}$ are compatible with both \mathcal{S} and \mathcal{S}' , and (2) (I, \mathcal{S}) satisfy the preconditions \mathcal{C}_{in} . We can now proceed with the semantics of procedures.

Definition 2. Let I be an instance over a schema \mathcal{S} . An instance I' over schema \mathcal{S}' is a possible outcome of applying P over the instance and schema (I, \mathcal{S}) if the following holds:

1. P is applicable on I .
2. $(I', \mathcal{S}') \models \mathcal{C}_{\text{out}}$.
3. The answers of the query $Q_{\mathcal{S} \setminus \text{Scope}}$ do not change: $Q_{\mathcal{S} \setminus \text{Scope}}(I) = Q_{\mathcal{S} \setminus \text{Scope}}(I')$.
4. The answers of each query Q in $\mathcal{Q}_{\text{safe}}$ over I are preserved: $Q(I) \subseteq Q(I')$.

In the definition, we state the schemas of instances I and I' explicitly, to reinforce the fact that schemas may change during the application of procedures. However, most of the time the schema can be understood from the instance, so we normally just say that an instance I' is a possible outcome of I (even if the schemas of I and I' are different). Let us also recall that we use $\text{Schema}(I)$ to denote the schema of an instance I .

Example 3 (Example 2 continued). Recall the procedure $P = (\text{Scope}, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{safe}})$ defined in Example 2. Consider the instance I over the schema \mathcal{S} with relations

EVisits and *LocVisits*, each with attributes *facility*, *patInsur*, and *timestp*, as shown in Figure 1 (a). Note first that P is indeed applicable on I . When applying the procedure P over I , we know from *Scope* that the only relation whose content can change is *LocVisits*, while *EVisits* (or more precisely, the projection of *EVisits* over *patInsur*, *facility* and *timestp*) is the same across all possible outcomes. Furthermore, we know from \mathcal{C}_{out} that in all possible outcomes the projection of *EVisits* over attributes *facility*, *patInsur*, and *timestp* must be the same as the projection of *LocVisits* over the same attributes. Finally, from $\mathcal{Q}_{\text{safe}}$ we know that the projection of *LocVisits* over these three attributes must be preserved.

Perhaps the most obvious possible outcome of applying P over I is that of the instance J_1 in Figure 1 (b), corresponding to the outcome where the tuple in *EVisits* that is not yet in *LocVisits* is migrated into this last relation. However, since we assume no control over the actions performed by the procedure P , it may well be that it is also migrating data from a different relation that we are not aware of, producing an outcome whose relation *EVisits* remains the same as in I and J_1 , but where *LocVisits* has additional tuples, as depicted in Figure 1 (c). Moreover, it may also be the case that the procedure alters the schema of *LocVisits*, adding an extra attribute *age*, importing the information from an unknown source, as shown in Figure 1 (d).

As we have seen in this example, in general the number of possible outcomes (and even the number of possible schemas) that result after a procedure is executed is infinite. For this reason, we are generally more interested in properties shared by all possible outcomes, which motivates the following definitions.

Definition 3. *The outcome set of applying a procedure P to I is defined as the set.*

$$\text{outcome}_P(I) = \{I' \mid I' \text{ is a possible outcome of applying } P \text{ to } I\}.$$
¹

4 Defining Common Database Tasks as Procedures

We now show additional examples of defining common database tasks as procedures within our framework. We show that data exchange, alter-table statements, and data cleaning can all be accommodated by the framework, and provide additional examples in Appendix A. It is worth noticing that in our first three examples we use only structure constraints, tgds, and egds as pre- and postconditions, and that our safe queries are all conjunctive queries. The last example calls for extending the language used to define procedures.

4.1 Data Exchange

We have already seen an example of specifying data-migration tasks as black-box procedures. However, a more detailed discussion will allow us to illustrate some

¹ Recall that the schema of instances I' is not necessarily the same as that of I .

of the basic properties of our framework. Following the notation introduced by Fagin et al. in [14], the most basic instance of the data-exchange problem considers a source schema \mathcal{S}_s , a target schema \mathcal{S}_t , and a set Σ of dependencies that define how data from the source schema are to be mapped to the target schema. The dependencies in Σ are usually tgds whose left-hand side is compatible with \mathcal{S}_s , and the right-hand side is compatible with \mathcal{S}_t . The data-exchange problem is as follows: Given a source instance I , compute a target instance J so that (I, J) satisfies all the dependencies in Σ . Instances J with this property are called *solutions* for I under Σ .

In encapsulating this task as a black box within our framework, we assume that the target and source schemas are part of the same schema. (Alternatively, one can define procedures working over different databases.) Let $(\mathcal{S}_s, \mathcal{S}_t, \Sigma)$ be as above. We construct the procedure $P^{st} = (\text{Scope}^{st}, \mathcal{C}_{\text{in}}^{st}, \mathcal{C}_{\text{out}}^{st}, \mathcal{Q}_{\text{safe}}^{st})$, where

- Scope^{st} contains an atom $R[*]$ for each relation R on the right-hand side of a tgd in Σ ;
- $\mathcal{C}_{\text{in}}^{st}$ contains a structure constraint $Q[A_1, \dots, A_n]$ for each query of the form $Q(A_1 : x_1, \dots, A_n : x_n)$ on the left-hand side of a tuple-generating dependency in Σ ;
- $\mathcal{C}_{\text{out}}^{st}$ is the set of all the tgds in Σ ; and
- $\mathcal{Q}_{\text{safe}}^{st}$ is the conjunction of all the atoms R appearing in any tgd in Σ .

By the semantics of procedures, it is not difficult to conclude that, for every pair of instances I and J over \mathcal{S}_s and \mathcal{S}_t , respectively, we have that J is a solution for I if and only if the instance $I \cup J$ over the schema $\mathcal{S}_s \cup \mathcal{S}_t$ is a possible outcome of applying P^{st} over $(I, \mathcal{S}_s \cup \mathcal{S}_t)$. We can make this statement much more general, as the set of all possible outcomes essentially corresponds to the set of solutions of the data-exchange setting.

Proposition 1. *An instance J over schema $\mathcal{S}_s \cup \mathcal{S}_t$ is a possible outcome of applying P^{st} over $(I, \mathcal{S}_s \cup \mathcal{S}_t)$ if and only if J is a solution for I under Σ .*

4.2 Alter Table Statements

In our framework, procedures can be defined to work over more than one schema, as long as the schemas satisfy the necessary input and compatibility conditions. This is inspired by SQL, where statements such as `INSERT INTO R (SELECT * FROM S)` would be executable over any schema, as long as the relations R and S have the same types of attributes in the same order. Thus, it seems logical to allow procedures that alter the schema of the existing database. To do so, we use structure constraints, as shown in the following example.

Example 4. Recall from Example 1 that, due to a change in the requirements, we now need to add the attribute *age* to the schema of *LocVisits*. In general, we capture *alter table* statements by procedures without scope, used only to alter the schema of the outcomes, so that it would satisfy the structural postconditions of procedures. In this case, we model a procedure that adds *age* to the schema

of *LocVisits* with the procedure $P' = (Scope', C'_{in}, C'_{out}, Q'_{safe})$, where $Scope'$ and Q'_{safe} are empty (if there is no scope, then the database does not change modulo adding attributes, so we do not include any safety guarantees), C'_{in} is the structure constraint *LocVisits*[*], stating that the relation exists in the schema, and C'_{out} is the structure constraint *LocVisits*[age], stating that *LocVisits* now has an age attribute. Note that the instance J_3 in Figure 1(d) with *EVisits* as in J_1 in Figure 1(b), is actually a possible outcome of applying P' over instance J_1 ; the part of the instance given by the schema of J_1 does not change, but we do add an extra attribute *age* to *LocVisits*, and we cannot really control the values of the newly added attribute.

We remark that the empty scope in P' guarantees that no relations or attributes are deleted when applying this procedure. This happens because $Q_{S \setminus Scope}$ must be compatible with the schema of all outcomes. However, nothing prevents us from adding extra attributes on top of *age*. This decision to use the open-world assumption on schemas reflects the understanding of procedures as black boxes, which we can execute but not control in other ways.

4.3 Data Cleaning

Data cleaning is a frequent and important task within database systems (see e.g., [15]). The most simple cleaning scenario one could envision is when we have a relation R whose attribute values are deemed incorrect or incomplete, and it is desirable to provide the correct values. There are, in general, multiple ways to do this; here we consider just a few of them.

The first possibility is to assume that we have the correct values in another relation, and to use this other relation to provide the correct values for R . Consider an example.

Example 5. Consider again the schema from Example 1. Recall that in Example 4 we added the attribute *age* to the schema of *LocVisits*. The problem is that we have no control over the newly added values of *age*. (If the procedure was a SQL alter-table statement, then the *age* column would be filled with nulls.) However, another relation, *Patients*, associates an *age* value with each pair of (*facility*, *patInsur*) values; all we need to do now is to copy the appropriate *age* value into each tuple in *LocVisits*. To this end, we specify the procedure $P^* = (Scope^*, C^*_{in}, C^*_{out}, Q^*_{safe})$, which copies the values of *age* from *Patients* into *LocVisits*, using the values of *facility* and *patInsur* as a reference.

*Scope**: We use the constraint *LocVisits*[age], so that the only piece of the database the procedure can alter is *age* in the relation *LocVisits*.

C^*_{in} : The preconditions are the structure constraints *LocVisits*[*facility*, *patInsur*, *age*] and *Patients*[*facility*, *patInsur*, *age*], plus the fact that the values of *facility* and *patInsur* need to determine the values of *age* in the *Patients* relation, specified with the dependency $Patients(facility : x, patInsur : y, age : z) \wedge Patients(facility : x, patInsur : y, age : w) \rightarrow z = w$. Note that in this case we do not actually need

the structure constraints in *Patients*, because they are implicit in the dependencies (they need to be compatible with the schema), but we keep them for clarity.

$\mathcal{C}_{\text{out}}^*$: The postcondition is the constraint $\text{Patients}(\text{facility} : x, \text{patInsur} : y, \text{age} : z) \wedge \text{LocVisits}(\text{facility} : x, \text{patInsur} : y, \text{age} : w) \rightarrow z = w$. Alternatively, if we know that all the $(\text{facility}, \text{patInsur})$ pairs from *Patients* are in *LocVisits* (which can be required with a precondition), we can specify the same postcondition via $\text{LocVisits}(\text{facility} : x, \text{patInsur} : y, \text{age} : w) \rightarrow \text{Patients}(\text{facility} : x, \text{patInsur} : y, \text{age} : w)$.

$\mathcal{Q}_{\text{safe}}^*$: Same as before, no guarantees are needed.

As desired, in all the outcomes of P^* the value of the *age* attribute in *LocVisits* is the same as in the corresponding tuple (if it exists) in *Patients* with the same *facility* and *patInsur* values. But then again, the procedure might modify the schema of some relations, or might even create auxiliary relations in the database in the process. What we gain is that this procedure will work regardless of the shape of relations *LocVisits* and *Patients*, as long as the schemas satisfy the compatibility and structure constraints.

In the above example we used a known auxiliary relation to clean the values of *age* in *LocVisits*. Alternatively, we could define a more general procedure that would, for instance, only remove nulls from *LocVisits*, without controlling which values end up replacing these nulls. In order to state this procedure, let us augment the language of tgds with an auxiliary predicate C (for *constant*) with a single attribute *val*, which is to take the role of the NOT NULL constraint in SQL: It is true only for the non-null values in D .

Example 6. Let us now define a procedure $\hat{P} = (\hat{Scope}, \hat{\mathcal{C}}_{\text{in}}, \hat{\mathcal{C}}_{\text{out}}, \hat{\mathcal{Q}}_{\text{safe}})$ that simply replaces all null values of the attribute *age* in relation *LocVisits* with non-null values.

\hat{Scope} : The scope is again $\text{LocVisits}[\text{age}]$, just as in the previous example.

$\hat{\mathcal{C}}_{\text{in}}$: In contrast with the procedure P^* of the previous example, this procedure is light on preconditions: We only need relation *LocVisits* to be present and have the *age* attribute.

$\hat{\mathcal{C}}_{\text{out}}$: The postcondition states that the attribute *age* of *LocVisits* no longer has null values. To express this, we use the auxiliary predicate C , and define the constraint $\text{LocVisits}(\text{age} : x) \rightarrow C(\text{val} : x)$, which states that no value in the attribute *age* in *LocVisits* is null.

$\hat{\mathcal{Q}}_{\text{safe}}$: Since we only want to eliminate null values, we also include the safety query $\text{LocVisits}(\text{age} : x, \text{facility} : y, \text{patInsur} : z) \wedge C(\text{val} : x)$, so that we preserve all the non-null values of *age* (with the correct *facility* and *patInsur* attached to these ages).

5 Basic Computational Tasks for Relational Procedures

In this section we study some formal properties of our procedure-centric framework, with the intent of showing how the proposed framework can be used as a

toolbox for reasoning about sequences of database procedures. We focus on what we call *relational procedures*, where the sets of pre- and postconditions are given by tgds, egds, or structure constraints, and safety queries can be conjunctive or total queries. While there clearly are interesting classes of procedures that do not fit into this special case in the proposed framework, we remark that relational procedures are general enough to account for a wide range of relational operations on data, including the examples in the previous section.

5.1 Applicability

In the proposed framework we focus on transformations of data sets given by sequences of procedures. Because we treat procedures as black boxes, the only description we have of the results of these transformations is that they ought to satisfy the output constraints of the procedures. In this situation, how can one guarantee that all the procedures will be applicable? Suppose that, for instance, we wish to apply procedures P_1 and P_2 to an instance I in sequential order: First P_1 , then P_2 . The problem is that, since output constraints do not fully determine the outcome of I after applying P_1 , we cannot immediately guarantee that this outcome is an instance that satisfies the preconditions of P_2 .

Given that the set of possible outcomes is in general infinite, our focus is on guaranteeing that *any* possible outcome of applying P_1 over I will satisfy the preconditions of P_2 . To formalize this intuition, we need to extend the notion of outcome to a set of instances. We define the outcome of applying a procedure P to a set of instances \mathcal{I} as

$$\text{outcome}_P(\mathcal{I}) = \bigcup_{I \in \mathcal{I}} \text{outcome}_P(I),$$

the union of the outcomes of all the instances in \mathcal{I} . Furthermore, for a sequence P_1, \dots, P_n of procedures we define the outcome of applying P_1, \dots, P_n to an instance I as the set

$$\text{outcome}_{P_1, \dots, P_n}(I) = \text{outcome}_{P_n}(\text{outcome}_{P_{n-1}}(\dots(\text{outcome}_{P_1}(I))\dots)).$$

We can now define the first problem of interest:

APPLICABILITY:

Input: A sequence P_1, \dots, P_n of procedures, a schema \mathcal{S} ;

Question: Is it true that, for any arbitrary instance I over \mathcal{S} , procedure P_n can be applied to each instance in the set $\text{outcome}_{P_1, \dots, P_{n-1}}(I)$?

It is not difficult to show that the APPLICABILITY problem is intimately related to the problem of implication of dependencies, defined as follows: Given a set Σ of dependencies and an additional dependency λ , is it true that all the instances that satisfy Σ also satisfy λ — that is, does Σ imply λ ? Indeed, consider a class \mathcal{L} of constraints for which the implication problem is known to be undecidable. Then one can easily show that the applicability problem is also undecidable for

those procedures whose pre- and postconditions are in \mathcal{L} : Intuitively, if we let P_1 be a procedure with a set Σ of postconditions, and P_2 a procedure with a dependency λ as a precondition, then it is not difficult to come up with proper scopes and safety queries so that $\text{outcome}_{P_1}(I)$ satisfies λ for every instance I over schema \mathcal{S} if and only if λ is true in all instances that satisfy Σ . However, as the following proposition shows, the applicability problem is undecidable already for very simple procedures, and even when we consider the *data-complexity* view of the problem, that is when we fix the procedure and take a particular input instance.

Proposition 2. *There are fixed procedures P_1 and P_2 that only use tgds for their constraints, and such that the following problem is undecidable. Given an instance I over schema \mathcal{S} , is it true that all the instances in $\text{outcome}_{P_1}(I)$ satisfy the preconditions of P_2 ?*

The proof of Proposition 2 is by reduction from the embedding problem for finite semigroups, shown to be undecidable in [20].

There are several lines of work aiming to identify practical classes of constraints for which the implication problem is decidable, and all that work can be applied in our framework. However, we opt for a stronger restriction: Since all of our examples so far use only structure constraints as preconditions, for the remainder of the paper we focus on procedures whose preconditions comprise structure constraints. In this setting, we have the following result.

Proposition 3. *APPLICABILITY is in polynomial time for sequences of relational procedures whose preconditions contain only structure constraints.*

5.2 Representing the Outcome Set

We have seen that deciding properties about the outcome set of a sequence of procedures (or even of a single procedure) can be a complicated task. One of the reasons is that procedures do not completely define their outcomes: We do not really know what will be the outcome of applying a sequence P_1, \dots, P_n of procedures to an instance I , we just know it will be an instance from the collection $\text{outcome}_{P_1, \dots, P_n}(I)$. This collection may well be of infinite size, but can it still be represented finitely? The database-theory community has developed multiple formalisms for representing sets of database instances, from notions of tables with incomplete information [16] to knowledge bases (see, e.g., [9]). In this section we study the possibility of representing outcomes of (sequences of) procedures by means of incomplete tables, along the lines of [16]. We also discuss some negative results about representing outcomes of general procedures in systems such as knowledge bases, but leave a more detailed study in this respect for future work.

The first observation we make is that allowing arbitrary tgds in procedures introduces problems with management of sequences of procedures. Essentially, any means of representing the outcome of a sequence of procedures needs to be so powerful that even deciding whether it is nonempty is going to be undecidable.

Proposition 4. *There is a fixed procedure P that does not use preconditions and only use tgds in their postconditions, such that the following problem is undecidable: Given an instance I , is the set $\text{outcome}_{P_1}(I)$ nonempty?*

The reason we view Proposition 4 as a negative result is because it rules out the possibility of using any “reasonable” representation system. Indeed, one would expect that deciding non-emptiness should be decidable in any reasonable way of representing infinite sets of instances. Proposition 4 is probably not surprising, since reasoning about tgds in general is known to be a hard problem. Perhaps more interestingly, in our case one can show that the above fact remains true even if one allows only *acyclic* tgds, which are arguably one of the most well-behaved classes of dependencies in the literature. The idea behind the proof is that one can simulate cyclic tgds via procedures with only acyclic tgds and no scope.

Example 7. Consider two procedures P_1 and P_2 , where $P_1 = (\text{Scope}^1, \mathcal{C}_{\text{in}}^1, \mathcal{C}_{\text{out}}^1, \mathcal{Q}_{\text{safe}}^1)$, with $\text{Scope}^1 = \{R[*], T[*]\}$, $\mathcal{C}_{\text{in}}^1 = \emptyset$, $\mathcal{C}_{\text{out}}^1 = \{R(A : x) \rightarrow T(A : x)\}$ and $\mathcal{Q}_{\text{safe}}^1 = R(A : x) \wedge T(A : x)$; P_2 has empty scope, preconditions, and safety queries, and has postconditions $\{T(A : x) \rightarrow R(A : x)\}$. Let I be an instance over the schema with relations R and T , both with attribute A . By definition, the set of possible outcomes of P_1 over I are all instances J that extend I and satisfy the dependency $R(A : x) \rightarrow T(A : x)$. However, the set $\text{outcome}_{P_1, P_2}(I)$ corresponds to all instances I' that extend I and satisfy both dependencies $R(A : x) \rightarrow T(A : x)$ and $T(A : x) \rightarrow R(A : x)$ (In other words, we can use P_2 to *filter out* all those instances J where $T^J \not\subseteq R^J$). Intuitively, this happens because the outcome set of applying P_2 over any instance not satisfying $T(A : x) \rightarrow R(A : x)$ is empty, and we define $\text{outcome}_{P_1, P_2}(I)$ as the union of each set $\text{outcome}_{P_2}(K)$, for each instance $K \in \text{outcome}_{P_1}(I)$.

By applying the idea of this example to the proof of Proposition 4, we show:

Proposition 5. *Proposition 4 holds for procedures P_1 and P_2 that only use acyclic tgds.*

Since acyclic tgds do not help, we may consider restrictions to full tgds. Still, even this is not enough for making the non-emptiness problem decidable, once one adds the possibility of having schema constraints in procedures.

Proposition 6. *There exists a sequence P_1, P_2, P_3 of procedures such that the following problem is undecidable: Given an instance I , is the set $\text{outcome}_{P_1, P_2, P_3}(I)$ nonempty? Here, all the procedures have no preconditions, and have postconditions built using acyclic sets of full tgds and schema constraints (and nothing else).*

Propositions 4 and 6 tell us that restricting the classes of dependencies allowed in procedures may not be enough to guarantee outcomes that can be represented by reasonable systems. Thus, we now adapt a different strategy: We restrict interplay between the postconditions of procedures, their scope, and

their safety queries. Let us define two important classes of procedures that will be used throughout this section.

We say that procedure $P = (Scope, \mathcal{C}_{in}, \mathcal{C}_{out}, \mathcal{Q}_{safe})$ is *safe scope* if the following holds:

- \mathcal{C}_{out} is a set of tgds where no relation in the right-hand side of a tgd appears also in the left-hand side of a tgd;
- The set $Scope$ contains exactly one constraint $R[*]$ for each relation R that appears on the right-hand side of a tgd in \mathcal{C}_{out} ; and
- The query \mathcal{Q}_{safe} corresponds to $\bigwedge_{R[*] \in Scope} R$, that is it binds precisely all the relations in the scope of P .

(For instance, procedure P in Example 2 is essentially a procedure with safe scope, as it can easily be transformed into one by slightly altering the safety query.)

We also define a class of procedures that ensure that certain attributes or relations be present in the schema. Formally, we say that a procedure $P = (Scope, \mathcal{C}_{in}, \mathcal{C}_{out}, \mathcal{Q}_{safe})$ is an *alter-schema procedure* if the following holds:

- Both $Scope$ and \mathcal{Q}_{safe} are empty; and
- \mathcal{C}_{out} is a set of structure constraints.

Let $\mathbb{P}^{safe, alter}$ be the class of all the procedures that are either safe scope or alter-schema procedures. The class $\mathbb{P}^{safe, alter}$ allows for practically-oriented interplay between migration and schema-alteration tasks and, as we will see in this section, is more manageable from the point of view of reasoning tasks, in terms of complexity. To begin with, deciding the non-emptiness of a sequence of procedures is essentially tractable for $\mathbb{P}^{safe, alter}$:

Theorem 1. *The problem of deciding, given an instance I and a sequence P_1, \dots, P_n of procedures in $\mathbb{P}^{safe, alter}$, whether $outcome_{P_1, \dots, P_n}(I) \neq \emptyset$, is in exponential time, and is polynomial if the number n of procedures is fixed.*

The proof of Theorem 1 is based on the idea of chasing instances with the dependencies in the procedures, and of adding attributes to schemas as dictated by the alter-schema procedures. As usual, to enable the chase we need to introduce labeled nulls in instances (see, e.g., [16, 14]), and composing procedures calls for extending the techniques of [2] to enable chase instances that already have null values. Using the enhanced approach, one can show that the result of the chase is a good over-approximation of the outcome of a sequence of procedures. To state this result, we introduce conditional tables [16].

Let \mathcal{N} be an infinite set of *null values* that is disjoint from the set of domain values D . A *naive instance* T over schema \mathcal{S} assigns a finite relation $R^T \subseteq (D \cup \mathcal{N})^n$ to each relation symbol R in \mathcal{S} of arity n . Conditional instances extend naive instances by attaching conditions over the tuples. Formally, an *element-condition* is a positive boolean combination of formulas of the form $x = y$ and $x \neq y$, where $x \in \mathcal{N}$ and $y \in (D \cup \mathcal{N})$. Then, a *conditional instance* T over schema \mathcal{S} assigns to each n -ary relation symbol R in \mathcal{S} a pair (R^T, ρ_R^T) , where

$R^T \subseteq (D \cup \mathcal{N})^n$ and ρ_R^T assigns an element-condition to each tuple $t \in R^T$. A conditional instance T is *positive* if none of the element-conditions in its tuples uses inequalities (of the form $x \neq y$).

To define the semantics, let $\text{Nulls}(T)$ be the set of all nulls in any tuple in T or in an element-condition used in T . Given a substitution $\nu : \text{Nulls}(T) \rightarrow D$, let ν^* be the extension of ν to a substitution $D \cup \text{Nulls}(T) \rightarrow D$ that is the identity on D . We say that ν *satisfies an element-condition* ψ , and write $\nu \models \psi$, if for every equality $x = y$ in ψ it is the case that $\nu^*(x) = \nu^*(y)$ and for every inequality $x \neq y$ we have that $\nu^*(x) \neq \nu^*(y)$. Furthermore, we define the set $\nu(R^T)$ as $\{\nu^*(t) \mid t \in R^T \text{ and } \nu \models \rho_R^T(t)\}$. Finally, for a conditional instance T , $\nu(T)$ is the instance that assigns $\nu(R^T)$ to each relation R in the schema.

The set of instances represented by T , denoted by $\text{rep}(T)$, is defined as $\text{rep}(T) = \{I \mid \text{there is a substitution } \nu \text{ such that } I \text{ extends } \nu(T)\}$. Note that the instances I in this definition could have potentially bigger schemas than $\nu(T)$, or, in other words, we consider the set $\text{rep}(T)$ to contain instances over any schema extending the schema of T .

The next result states that conditional instances are good over-approximations for the outcomes of sequences of procedures. More interestingly, these approximations preserve the *minimal instances* of outcomes. To put this formally, we say that an instance J in a set \mathcal{I} of instances is *minimal* if there is no instance $J' \in \mathcal{I}, J' \neq J$, and such that J extends J' .

Proposition 7. *Let I be an instance and P_1, \dots, P_n be a sequence of procedures in $\mathbb{P}^{\text{safe}, \text{alter}}$. Then either $\text{outcome}_{P_1, \dots, P_n} = \emptyset$ or one can construct, in exponential time (or polynomial if n is fixed), a conditional instance T such that*

- $\text{outcome}_{P_1, \dots, P_n}(I) \subseteq \text{rep}(T)$; and
- If J is a minimal instance in $\text{rep}(T)$, then J is also minimal in $\text{outcome}_{P_1, \dots, P_n}(I)$.

We remark that this proposition can be extended to include procedures defined only with egds, at the cost of a much more technical presentation. While having an approximation with these properties is useful for reasoning tasks related to CQ answering, or in general checking any criterion that is closed under extensions of instances, there is still the question of whether one can find any reasonable class of properties whose entire outcomes can be represented by these tables. However, as the following example shows, this does not appear to be possible, unless one is restricted to sequences of procedures almost without interaction with each other (see an example in appendix A.2).

Example 8. Consider a procedure $P = (\text{Scope}, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{safe}})$ with safe scope, where $\text{Scope} = S[*]$, \mathcal{C}_{in} is empty, $\mathcal{C}_{\text{out}} = \{R(A : x) \rightarrow S(A : x)\}$ and $\mathcal{Q}_{\text{safe}} = S$. Consider now the conditional instance T over the schema with relations R and S , both with attribute A , given by $R^T = \{1, 2\}$ and $S^T = \{1, 2\}$. One could be tempted to say that T is itself a representation of the set $\text{outcome}_P(\text{rep}(T))$, and indeed $\text{rep}(T)$ and $\text{outcome}_P(\text{rep}(T))$ share their only minimal instance (essentially, the instance given by T). However, the open-world assumption behind $\text{rep}(T)$ allows for instances that do not satisfy \mathcal{C}_{out} , whereas all outcomes in

$outcome_P(rep(T))$ must satisfy \mathcal{C}_{out} . One can in fact generalize this argument to show that conditional instances are not enough to fully represent outcome sets.

Example 8 suggest that one could perhaps combine conditional instances with a knowledge base, to allow for a complete representation of the outcome set of sequences of safe procedures. However, this would require studying the interplay of these two different types of representation systems, a line of work which is interesting in its own right.

6 Future Work and Opportunities

In this paper, we introduced basic building blocks for a proposed framework for assessing achievability of data-quality constraints. We demonstrated that the framework is general enough to represent nontrivial database tasks, and exhibited realistic classes of procedures for which reasoning tasks can be tractable. Our next step is to address the problem of assessing achievability of constraints, which can be formalized as follows. Let Q be a boolean query, Π a set of procedures, and I an instance over a schema \mathcal{S} . Then we say that I can be readied for Q using Π if there is a sequence P_1, \dots, P_n of procedures (possibly empty and possibly with repetitions) from Π such that Q is compatible with and true in each instance I' in the set $outcome_{P_1, \dots, P_n}(I)$. (If the latter conditions involving Q are true on I , then we say that I is ready for Q .) We are confident that this problem is decidable for sets of procedures in $\mathbb{P}^{safe, alter}$, and we plan on looking into more expressive fragments.

The proposed framework presents opportunities for several directions of further research. One line of work would involve understanding how to represent outcomes of sequences of procedures, or how to obtain good approximations of outcomes of more expressive classes of procedures. To solve this problem, we would need a better understanding of the interplay between conditional tables and knowledge bases, which would be interesting in its own right.

We also believe that our framework is general enough to allow reasoning on other data paradigms, or even across various different data paradigms. Our black-box abstraction could, for example, offer an effective way to reason about procedures involving unstructured text data, or even data transformations using machine-learning tools, as long as one can obtain some guarantees on the data outcomes of these tools.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases: the logical level*. Addison-Wesley Longman Publishing Co., Inc., 1995.
2. M. Arenas, J. Pérez, and J. Reutter. Data exchange beyond complete data. *Journal of the ACM*, 60(4):28, 2013.
3. D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, M. Lenzerini, and M. Mecella. Modeling data & processes for service specifications in Colombo. In *Proceedings of the Open Interop. Workshop on Enterprise Modelling and Ontologies for Interoperability*, 2005.

4. D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of web services in Colombo. In *Proceedings of the Thirteenth Italian Symposium on Advanced Database Systems (SEBD)*, pages 8–15, 2005.
5. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic service composition based on behavioral descriptions. *Int. J. Cooperative Inf. Syst.*, 14(4):333–376, 2005.
6. M. Bergman, T. Milo, S. Novgorodov, and W. Tan. QOCO: A query oriented data cleaning system with oracles. *PVLDB*, 8(12):1900–1911, 2015.
7. M. Bergman, T. Milo, S. Novgorodov, and W. C. Tan. Query-oriented data cleaning with oracles. In *Proceedings of ACM SIGMOD*, pages 1199–1214, 2015.
8. K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards formal analysis of artifact-centric business process models. In *International Conference on Business Process Management*, pages 288–304. Springer, 2007.
9. M. Bienvenu and M. Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Reasoning Web International Summer School*, pages 218–307. Springer, 2015.
10. I. Chengalur-Smith and H. Pazer. Decision complacency, consensus and consistency in the presence of data quality information. In *Information Quality*, pages 88–101, 1998.
11. D. Deutch and T. Milo. *Business Processes: A Database Perspective*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
12. A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *Proceedings of the 12th International Conference on Database Theory*, pages 252–267. ACM, 2009.
13. B. Devlin. *Data Warehouse: From Architecture to Implementation*. Addison-Wesley Longman, 1996.
14. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.
15. W. Fan and F. Geerts. *Foundations of Data Quality Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
16. T. Imieliński and W. Lipski Jr. Incomplete information in relational databases. *Journal of the ACM (JACM)*, 31(4):761–791, 1984.
17. B. Kahn, D. Strong, and R. Wang. Information quality benchmarks: Product and service performance. *Comm. ACM*, 45(4ve):184–192, 2002.
18. R. Kimball and J. Caserta. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Wiley, 2004.
19. D. Kindig and G. Stoddart. What is population health? *Am. J. Public Health*, 93(3):380–383, 2003.
20. P. G. Kolaitis, J. Panttaja, and W.-C. Tan. The complexity of data exchange. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 30–39, 2006.
21. S. Krishnan, J. Wang, M. J. Franklin, K. Goldberg, T. Kraska, T. Milo, and E. Wu. SampleClean: Fast and reliable analytics on dirty data. *IEEE Data Eng. Bull.*, 38(3):59–75, 2015.
22. Y. Lee, L. Pipino, R. Wang, and J. Funk. *Journey to Data Quality*. MIT Press, 2009.
23. Y. Lee and D. Strong. Knowing-why about data processes and data quality. *Journal of Management Information Systems*, 20(3):13–39, 2004.
24. Y. Lee, D. Strong, B. Kahn, and R. Wang. AIMQ: a methodology for information quality assessment. *Information & Management*, 40:133–146, 2002.

25. A. McAlearney. *Population Health Management: Strategies to Improve Outcomes*. Health Administration Press, 2003.
26. W. Nutt, S. Paramonov, and O. Savkovic. Implementing query completeness reasoning. In *ACM CIKM*, pages 733–742, 2015.
27. Premier, Inc.: Alliance of healthcare providers on a mission to transform healthcare, 2016. <https://www.premierinc.com>.
28. S. Razniewski, F. Korn, W. Nutt, and D. Srivastava. Identifying the extent of completeness of query answers over partially complete databases. In *ACM SIGMOD*, pages 561–576, 2015.
29. R. Wang. A product perspective on total data quality management. *Comm. ACM*, 41(2), 1998.
30. R. Wang, Y. Lee, L. Pipino, and D. Strong. Manage your information as product: The keystone to quality information. *MIT Sloan Management Review*, 39(4):95–105, 1998.
31. R. Wang and D. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of Management Information Systems*, 12(4):5–34, 1996.
32. T. Young. *Population Health: Concepts and Methods*. Oxford University Press, 1998.

A Additional Examples

A.1 SQL data-modification statements

We show how to encode arbitrary SQL INSERT and DELETE statements as procedures. Due to dealing with arbitrary SQL, we relax the constraints and queries that we use.

INSERT statements: Consider a SQL statement of the form `INSERT INTO S Q`, where Q is a relational-algebra query.

Scope: Not surprisingly, the scope of the procedure is the relation S .

C_{in} : The precondition for the procedure is that all the relation names and attributes mentioned in Q must be present in the database.

C_{out} : The postcondition is stated using the constraint $Q \subseteq S$. (Note that the SQL statement only works when Q and S have the same arity.)

Q_{safe} : Since we are inserting tuples, we need the query S to be preserved.

Alternatively, we can specify an INSERT statement of the form `INSERT INTO S VALUES \bar{a}` , with \bar{a} a tuple of values. In order to formalize this, we just need to change the postcondition of the procedure to $\bar{a} \subseteq S$.

DELETE statements: Consider a SQL statement of the form `DELETE FROM S WHERE C` , in which C is a boolean combination of conditions.

Scope: The scope is the relation S , as expected.

C_{in} : The precondition for the procedure is that all the relations and attributes mentioned in C must be present in the database.

C_{out} : There are no postconditions in this query.

Q_{safe} : Let Q_C be the query `SELECT * FROM S WHERE C` . Then the safety query is $S - Q_C$, which preserves only those tuples that are not to be deleted.

A.2 Representing sequences of procedures

As we mentioned, one possibility to obtain a full representation of sequences of procedures is to further restrict the scope of sequences of safe procedures. To be more precise, let us say that a sequence P_1, \dots, P_n of procedures is a *safe sequence* if (1) each P_i is either an alter-schema procedure or a safe-scope procedure that only uses tgds, and (2) for every $1 \leq j \leq n$, none of the atoms on the right-hand side of a tgd in P_j is part of the scope of any P_i , with $i \leq j$. Intuitively, safe sequences of procedures restrict the possibility of sequencing data-migration tasks when the result of one migration is used as an input for the next one.

A conditional instance *with scope* is a pair $\mathbb{T} = (T, Rel)$, where T is a conditional instance and Rel is a set of relation names. The set of instances represented by \mathbb{T} , denoted again by $rep(\mathbb{T})$, now contains all the instances J in $rep(T)$ where, for each relation $R \in \text{Schema}(\nu(T))$ that is not in Rel , the projection of R^J over the attributes of R in T is the same as $R^{\nu(T)}$. (In other words, we allow extra tuples only in the relations whose symbols are in the set Rel .) It is now not difficult to show the following result.

Proposition 8. *For each instance I and each safe sequence P_1, \dots, P_n of procedures one can construct a conditional instance \mathbb{T} with scope such that $\text{rep}(\mathbb{T}) = \text{outcome}_{P_1, \dots, P_n}(I)$.*

B Proofs and Intermediate Results

B.1 Proof of Proposition 2

The reduction is from the complement of the embedding problem for finite semigroups, shown to be undecidable in [20], and it is itself an adaptation of the proof of Theorem 7.2 in [2]. Note that, since we do not intend to add attributes nor relations in the procedures of this proof, we can drop the named definition of queries, treating CQs now as normal conjunctions of relational atoms.

The embedding problem for finite semigroups problem can be stated as follows. Consider a pair $\mathbf{A} = (A, g)$, where A is a finite set and $g : A \times A \rightarrow A$ is a partial associative function. We say that \mathbf{A} is embeddable in a finite semigroup if there exists $\mathbf{B} = (B, f)$ such that $A \subseteq B$ and $f : B \times B \rightarrow B$ is a total associative function. The embedding problem for finite semigroups is to decide whether an arbitrary $\mathbf{A} = (A, g)$ is embeddable in a finite semigroup.

Consider the schema $\mathcal{S} = \{C(\cdot, \cdot), E(\cdot, \cdot), N(\cdot, \cdot), G(\cdot, \cdot, \cdot), F(\cdot), D(\cdot)\}$. The idea of the proof is as follows. We use relation G to encode binary functions, so that a tuple (a, b, c) in G intuitively corresponds to saying that $g(a, b) = c$, for a function g . Using our procedure we shall mandate that the binary function encoded in G is total and associative. We then encode $\mathbf{A} = (A, g)$ into our input instance I : the procedure will then try to embed A into a semigroup whose function is total.

In order to construct the procedures, we first specify the following set Σ of tgds. First we add to Σ a set of dependencies ensuring that all elements in the relation G are collected into D :

$$G(x, u, v) \rightarrow D(x) \tag{1}$$

$$G(u, x, v) \rightarrow D(x) \tag{2}$$

$$G(u, v, x) \rightarrow D(x) \tag{3}$$

The next set verifies that G is total and associative:

$$D(x) \wedge D(y) \rightarrow \exists z G(x, y, z) \tag{4}$$

$$G(x, y, u) \wedge G(u, z, v) \wedge G(y, z, w) \rightarrow G(x, w, v) \tag{5}$$

Next we include dependencies that are intended to force relation E to be an equivalence relation over all elements in the domain of G .

$$D(x) \rightarrow E(x, x) \tag{6}$$

$$E(x, y) \rightarrow E(y, x) \tag{7}$$

$$E(x, y) \wedge E(y, z) \rightarrow E(x, z) \tag{8}$$

The next set of dependencies we add Σ ensure that G represents a function that is consistent with the equivalence relation E .

$$G(x, y, z) \wedge E(x, x') \wedge E(y, y') \wedge E(z, z') \rightarrow G(x', y', z') \quad (9)$$

$$G(x, y, z) \wedge G(x', y', z') \wedge E(x, x') \wedge E(y, y') \rightarrow E(z, z') \quad (10)$$

The final tgtd in Σ serves us to collect possible errors when trying to embed $\mathbf{A} = (A, g)$. The intuition for this tgtd will be made clear once we outline the reduction, but the idea is to state that the relation F now contains everything that is in R , as long as a certain property holds on relations E , C and N .

$$E(x, y) \wedge C(u, x) \wedge C(v, y) \wedge N(u, v) \wedge R(w) \rightarrow F(w) \quad (11)$$

Let then Σ consists of tgds (1)-(11). We construct fixed procedures $P_1 = (Scope^1, \mathcal{C}_{in}^1, \mathcal{C}_{out}^1, \mathcal{Q}_{safe}^1)$ and $P_2 = (Scope^2, \mathcal{C}_{in}^2, \mathcal{C}_{out}^2, \mathcal{Q}_{safe}^2)$ as follows.

procedure P_1 :

$Scope^1$: The scope of P_1 consists of relations G , E , D and F , which corresponds to the constraints $\{G[*], E[*], D[*], F[*]\}$.

\mathcal{C}_{in}^1 : There are no preconditions for this procedure.

\mathcal{C}_{out}^1 : The postconditions are the tgds in Σ .

\mathcal{Q}_{safe}^1 : This query ensures that no information is deleted from all of G , E and F (and thus that no attributes are added to them): $G(x, y, z) \wedge E(u, v) \wedge D(w) \wedge F(p)$.

procedure P_2 :

$Scope^2$: The scope of P_2 is empty.

\mathcal{C}_{in}^2 : The precondition for this constraint is $R(x) \rightarrow F(x)$.

\mathcal{C}_{out}^2 : There are no postconditions.

\mathcal{Q}_{safe}^2 : There is no safety query.

Note that P_2 does not really do anything, it is only there to check that R is contained in F . We can now state the reduction. On input $\mathbf{A} = (A, g)$, where $A = \{a_1, \dots, a_n\}$, we construct an instance $I_{\mathbf{A}}$ given by the following interpretations:

- $E^{I_{\mathbf{A}}}$ contains the pair (a_i, a_i) for each $1 \leq i \leq n$ (that is, for each element of A);
- $G^{I_{\mathbf{A}}}$ contains the triple (a_i, a_j, a_k) for each $a_i, a_j, a_k \in A$ such that $g(a_i, a_j) = a_k$;
- $D^{I_{\mathbf{A}}}$ and $F^{I_{\mathbf{A}}}$ are empty, while $R^{I_{\mathbf{A}}}$ contains a single element d not in A ;
- $C^{I_{\mathbf{A}}}$ contains the pair (i, a_i) for each $1 \leq i \leq n$; and
- $N^{I_{\mathbf{A}}}$ contains the pair (i, j) for each $i \neq j$, $1 \leq i \leq n$ and $1 \leq j \leq n$.

Let us now show $\mathbf{A} = (A, g)$ is embeddable in a finite semigroup if and only if $outcome_{P_1}(I)$ contains an instance I' such that I' does not satisfy the precondition $R(x) \rightarrow F(x)$ of procedure P_2 .

(\implies) Assume that $\mathbf{A} = (A, g)$ is embeddable in a finite semigroup, say the semigroup $\mathbf{B} = (B, f)$, where f is total. Let J be the instance such that E^J is the identity over B , $D^J = B$ and G^J contains a pair (b_1, b_2, b_3) if and only if $f(b_1, b_2) = b_3$; F^J is empty and relations N , C and R are interpreted as in $I_{\mathbf{A}}$. It is easy to see that $J \models \Sigma$, $Q_{S \setminus \text{scope}}$ is preserved and that $\mathcal{Q}_{\text{safe}}(I_{\mathbf{A}}) \subseteq \mathcal{Q}_{\text{safe}}(J)$, this last because \mathbf{A} was said to be embeddable in \mathbf{B} . We have that J then does belong to $\text{outcome}_{P_1}(I)$, but J does not satisfy the constraint $R(x) \rightarrow F(x)$.

(\impliedby) Assume now that there is an instance $J \in \text{outcome}_{P_1}(I)$ that does not satisfy $R(x) \rightarrow F(x)$. Note that, because of the scope of P_1 , the interpretation of C , N and R of J must be just as in I . Thus it must be that the element d is not in F^J , because it is the only element in R^J . Construct a finite semigroup $\mathbf{B} = (B, f)$ as follows. Let B consists of one representative of each equivalence class in E^J , with the additional restriction that each a_i in A must be picked as its representative. Further, define $f(b_1, b_2) = b_3$ if and only if $G(b_1, b_2, b_3)$ is in G . Note that J satisfies the tgds in Σ , in particular G is associative and E acts as an equivalence relation over G , which means that f is indeed associative, total, and well defined. It remains to show that \mathbf{A} can be embedded in \mathbf{B} , but since G^J and E^J are supersets of $G^{I_{\mathbf{A}}}$ and $E^{I_{\mathbf{A}}}$ (because of the safety query of P_1), all we need to show is that each a_i is in a separate equivalence relation. But this hold because of tgd (11) in Σ : if two elements from A are in the same equivalence relation then the left hand side of (11) would hold in $I_{\mathbf{A}}$, which contradicts the fact that F^J does not contain d .

B.2 Proof of Proposition 3

Let $P = (\text{Scope}, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{safe}})$. We first show how to construct, for each instance I over a schema \mathcal{S} , the *minimal* schema \mathcal{S}_{min} such that all pairs (J, \mathcal{S}') that are possible outcomes of applying P over (I, \mathcal{S}) are such that \mathcal{S}' extend \mathcal{S}_{min} .

The algorithm receives a procedure P and a schema \mathcal{S} and outputs either \mathcal{S}_{min} , if the procedure is applicable, or a failure signal in case there is no schema satisfying the output constraints of the procedure. Along the algorithm we will be assigning numbers to some of the relations in \mathcal{S}_{min} . This is important to be able to decide failure.

Algorithm $A(P, \mathcal{S})$ for constructing \mathcal{S}_{min}

Input: procedure $P = (\text{Scope}, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{safe}}$ and schema \mathcal{S} .

Output: either *failure* or a schema \mathcal{S}_{min} .

1. If \mathcal{S} does not satisfy the structural constraints in \mathcal{C}_{in} or is not compatible with either $\mathcal{Q}_{\text{safe}}$ or $\mathcal{Q}_{S \setminus \text{scope}}$, output failure. Otherwise, continue.
2. Start with $\mathcal{S}_{\text{min}} = \emptyset$.
3. For each total query R in $\mathcal{Q}_{\text{safe}}$, assume that $|\mathcal{S}(R)| = k$. Set $\mathcal{S}_{\text{min}}(R) = \mathcal{S}(R)$, and label R with k .
4. Add to \mathcal{S}_{min} all relations R mentioned in an atom $R[*]$ in \mathcal{C}_{out} (if they are not already part of \mathcal{S}_{min}), without associating any attributes to them

5. In the following instructions we construct a set $\Gamma(P, \mathcal{S})$ of pairs of relations and attributes. Intuitively, a pair $(R, \{a_1, \dots, a_n\})$ in $\Gamma(P, \mathcal{S})$ states that each schema in the output of P must contain a relation R with attributes a_1, \dots, a_n .
 - For each relation R in \mathcal{S} that is not mentioned in Scope , add to $\Gamma(P, \mathcal{S})$ the pair $(R, \mathcal{S}(R))$.
 - For each constraint $R[a_1, \dots, a_n]$ in Scope , add the pair $(R, \mathcal{S}(R) \setminus \{a_1, \dots, a_n\})$ to $\Gamma(P, \mathcal{S})$.
 - For each atom $R(a_1 : x_1, \dots, a_n : x_n)$ in $\mathcal{Q}_{\text{safe}}$ add to $\Gamma(P, \mathcal{S})$ the pair $(R, \{a_1, \dots, a_n\})$.
 - For each atom $R(a_1 : x_1, \dots, a_n : x_n)$ in a tgd or egd in \mathcal{C}_{out} add to $\Gamma(P, \mathcal{S})$ the pair $(R, \{a_1, \dots, a_n\})$.
 - For each constraint $R[a_1, \dots, a_n]$ in \mathcal{C}_{out} , add to $\Gamma(P, \mathcal{S})$ the pair $(R, \{a_1, \dots, a_n\})$.
6. For each pair (R, A) in $\Gamma(P, \mathcal{S})$, do the following.
 - If R is not yet in \mathcal{S}_{\min} , add R to \mathcal{S}_{\min} and set $\mathcal{S}_{\min}(R) = A$;
 - If R is in \mathcal{S}_{\min} , update $\mathcal{S}_{\min}(R) = \mathcal{S}_{\min}(R) \cup A$.
7. If \mathcal{S}_{\min} contains a relation R labelled with a number n where, $\mathcal{S}_{\min}(R) > n$, output failure. Otherwise output \mathcal{S}_{\min} .

By direct inspection of the algorithm, we can state the following.

Proposition 9. *Let $P = (\text{Scope}, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{safe}})$ be a relational procedure and \mathcal{S} a relational schema. Then for each relation R in \mathcal{S}_{\min} with attributes $\{a_1, \dots, a_n\}$, every instance I over \mathcal{S} and every pair (J, \mathcal{S}') in the outcome of applying P to (I) , we have that $\mathcal{S}(R)$ is defined, with $\{a_1, \dots, a_n\} \subseteq \mathcal{S}(R)$.*

Furthermore, the following lemma specifies, in a sense, the correctness of the algorithm.

Lemma 1. *Let $P = (\text{Scope}, \mathcal{C}_{\text{in}}, \mathcal{C}_{\text{out}}, \mathcal{Q}_{\text{safe}})$ be a relational procedure and \mathcal{S} a relational schema. Then:*

- i) *If $A(P, \mathcal{S})$ outputs failure, either P cannot be applied over any instance I over \mathcal{S} , or for each instance I over \mathcal{S} the set $\text{outcome}_P(I)$ is empty.*
- ii) *If $A(P, \mathcal{S})$ outputs \mathcal{S}_{\min} , then the schema of any instance in $\text{outcome}_P(I)$ extends \mathcal{S}_{\min} .*

Proof. For i), not that if some of the components of P are not compatible with \mathcal{S} , or \mathcal{S} does not satisfy the constraints in \mathcal{C}_{in} , then clearly P cannot be applied over any instance I over \mathcal{S} . Assume then that \mathcal{S} satisfies all compatibilities and preconditions in P , but $A(P, \mathcal{S})$ outputs failure. Then \mathcal{S}_{\min} contains a relation R such that $|\mathcal{S}_{\min}(R)| = m$, but R is labelled with number k , for $k < \ell$. From the algorithm, we this implies that $|\mathcal{S}_{\min}(R)| > |\mathcal{S}(R)|$, but that there is a query R in $\mathcal{Q}_{\text{safe}}$. Clearly, $\mathcal{Q}_{\text{safe}}$ cannot be preserved under any outcome, since by Observation 9 we require the schemas of outcomes to assign more attributes to R than those assigned by \mathcal{S}_{\min} , and thus the cardinality of tuples in the answer of R differs between I and its possible outcomes. Finally, item ii) is a direct consequence of Observation 9.

Note that the algorithm (A, P) runs in polynomial time, and that the total size of \mathcal{S}_{\min} (measured as the number of relations and attributes) is at most the size of \mathcal{S} and P combined. Thus, to decide the applicability problem for a sequence P_1, \dots, P_n of procedures, all we need to do is to perform subsequent calls to the algorithm, setting $\mathcal{S}_0 = \mathcal{S}$ and then using $\mathcal{S}_i = A(P_i, \mathcal{S}_{i-1})$ as the input for the next procedures. If $A(P_n, \mathcal{S}_{n-1})$ outputs a schema, then the answer to the applicability problem is affirmative, otherwise if some call to $A(P_i, \mathcal{S}_i - 1)$ outputs failure, the answer is negative.

B.3 Proof of Proposition 4

This proof is a simple adaptation of the reduction we used in the proof of Proposition 2. Indeed, consider again the schema \mathcal{S} from this proof, and the procedure P given by:

Scope: The scope of P consists of relations G , E , D and F , which corresponds to the constraints $G[*]$, $E[*]$, $D[*]$ and $F[*]$.

\mathcal{C}_{in} : There are no preconditions for this procedure.

\mathcal{C}_{out} : The postconditions are the tgds in Σ plus the tgd $F(x) \rightarrow R(x)$.

$\mathcal{Q}_{\text{safe}}$: This query ensures that no information is deleted from all of G , E and F : $G(x, y, z) \wedge E(u, v) \wedge D(w) \wedge F(p)$.

Given a finite semigroup \mathbf{A} , we construct now the following instance I :

- $E^{I\mathbf{A}}$ contains the pair (a_i, a_i) for each $1 \leq i \leq n$ (that is, for each element of A);
- $G^{I\mathbf{A}}$ contains the triple (a_i, a_j, a_k) for each $a_i, a_j, a_k \in A$ such that $g(a_i, a_j) = a_k$;
- All of $D^{I\mathbf{A}}$, $F^{I\mathbf{A}}$ and $R^{I\mathbf{A}}$ are empty;
- $C^{I\mathbf{A}}$ contains the pair (i, a_i) for each $1 \leq i \leq n$; and
- $N^{I\mathbf{A}}$ contains the pair (i, j) for each $i \neq j$, $1 \leq i \leq n$ and $1 \leq j \leq n$.

By a similar argument as the one used in the proof of Proposition 2, one can show that $\text{outcome}_P(I)$ has an instance if and only if \mathbf{A} is embeddable in a finite semigroup. The intuition is that now we are adding the constraint $F(x) \rightarrow R(x)$ as a postcondition, and since R is not part of the scope of the procedure the only way to satisfy this restriction is if we do not fire the tgd (11) of the set Σ constructed in the aforementioned proof. This, in turn, can only happen if \mathbf{A} is embeddable.

B.4 Proof of Proposition 6

The reduction, just as that of Proposition 2, is by reduction from the embedding problem for finite semigroups, and builds up from this proposition. Let us start by defining the procedures P_1 , P_2 and P_3 .

For procedure P_1 we first build a set I_1 of tgs. This set is similar to the set Σ used in Proposition 2, but using three additional *dummy* relations G^d , E^d and G^{binary} .

First we add to I_1 dependencies that collect elements of G into D , and that initialize E as a reflexive relation.

$$\begin{aligned} G(x, u, v) &\rightarrow D(x) \\ G(u, x, v) &\rightarrow D(x) \\ G(u, v, x) &\rightarrow D(x) \\ D(x) &\rightarrow E(x, x) \end{aligned}$$

Next the dependency that states that F contains everything in R if some conditions about E occur.

$$E(x, y) \wedge C(u, x) \wedge C(v, y) \wedge N(u, v) \wedge R(w) \rightarrow F(w) \quad (12)$$

The dependencies that assured that E was an equivalence relation where acyclic, so we replace the right hand side with a dummy relation.

$$\begin{aligned} E(x, y) &\rightarrow E^d(y, x) \\ E(x, y) \wedge E(y, z) &\rightarrow E^d(x, z) \end{aligned}$$

Next come the dependencies assuring G is a total and associative function, using also dummy relations.

$$\begin{aligned} D(x) \wedge D(y) &\rightarrow \exists z G^{\text{binary}}(x, y) \\ G(x, y, u) \wedge G(u, z, v) \wedge G(y, z, w) &\rightarrow G^d(x, w, v) \end{aligned}$$

Finally, the dependencies that were supposed to ensure that E worked as the equality over function G , using again the dummy relations.

$$\begin{aligned} G(x, y, z) \wedge E(x, x') \wedge E(y, y') \wedge E(z, z') &\rightarrow G^d(x', y', z') \\ G(x, y, z) \wedge G(x', y', z') \wedge E(x, x') \wedge E(y, y') &\rightarrow E^d(z, z') \end{aligned}$$

We can now define procedure P_1 :

Scope: The scope of P_1 consists of relations G, E, D, F, G^d, E^d and G^{binary} which corresponds to the constraints $G[*], E[*], D[*], F[*], E^d[*], G^d[*]$ and $G^{\text{binary}}[*]$.

\mathcal{C}_{in} : There are no preconditions for this procedure.

\mathcal{C}_{out} : The postconditions are the tgds in I_1 .

$\mathcal{Q}_{\text{safe}}$: This query ensures that no information is deleted from all of G, E, F, G^d, E^d and G^{binary} : $G(x, y, z) \wedge E(u, v) \wedge D(w) \wedge F(p) \wedge G^d(x', y', z') \wedge E^d(u', v') \wedge G^{\text{binary}}(a, b)$.

Note that, even though relations G and E are not mentioned in the right hand side of any tgd in I_1 , they are part of the scope and thus they could be modified by the procedures P_1 .

The procedure P_2 has no scope, no safety queries, no precondition, and the only postcondition is the presence of a third attribute, say C , in G^{binary} , by using a structural constraint $G^{\text{binary}}[A, B, C]$ (to maintain consistency with our unnamed perspective, we assume that these three attributes are ordered $A <_{\mathcal{A}} B <_{\mathcal{A}} C$).

To define the final procedure, consider the following set of tgds Γ_3 .

$$\begin{aligned} E^d(x, y) &\rightarrow E(x, y) \\ G^d(x, y, z) &\rightarrow G(x, y, z) \\ G^{\text{binary}}(x, y, z) &\rightarrow G(x, y, z) \\ F(x) &\rightarrow F^{\text{check}}(x) \end{aligned}$$

Then we define procedure P_3 is as follows.

Scope: The scope of P_3 is again empty.

\mathcal{C}_{in} : There are no preconditions for this procedure.

\mathcal{C}_{out} : The postconditions are the tgds in Γ_3 .

$\mathcal{Q}_{\text{safe}}$: There are also no safety queries for this procedure.

Let \mathcal{S} be the schema containing relations $G, E, D, F, F^{\text{check}}, G^d, E^d$ and G^{binary} and R . The attribute names are of no importance for this proof, except for G^{binary} , which associates attributes A and B .

Given a finite semigroup \mathbf{A} , we construct now the following instance $I_{\mathbf{A}}$:

- $E^{I_{\mathbf{A}}}$ contains the pair (a_i, a_i) for each $1 \leq i \leq n$ (that is, for each element of A);
- $G^{I_{\mathbf{A}}}$ contains the triple (a_i, a_j, a_k) for each $a_i, a_j, a_k \in A$ such that $g(a_i, a_j) = a_k$;
- All of $D^{I_{\mathbf{A}}}, F^{I_{\mathbf{A}}}$ and $F^{\text{check}I_{\mathbf{A}}}$ are empty;
- $R^{I_{\mathbf{A}}}$ has a single element d not used elsewhere in $I_{\mathbf{A}}$
- $C^{I_{\mathbf{A}}}$ contains the pair (i, a_i) for each $1 \leq i \leq n$; and
- $N^{I_{\mathbf{A}}}$ contains the pair (i, j) for each $i \neq j, 1 \leq i \leq n$ and $1 \leq j \leq n$.

Let us now show $\mathbf{A} = (A, g)$ is embeddable in a finite semigroup if and only if $\text{outcome}_{P_1, P_2, P_3}(I)$ is nonempty.

(\implies) Assume that $\mathbf{A} = (A, g)$ is embeddable in a finite semigroup, say the semigroup $\mathbf{B} = (B, f)$, where f is total. Let J be the instance over \mathcal{S} such that both E^{dJ} and E^J are the identity over B , $D^J = B$, both G^{dJ} and G^J contains a pair (b_1, b_2, b_3) if and only if $f(b_1, b_2) = b_3$; $G^{\text{binary}J}$ is the projection of G^J over its two first attributes, F^J and $F^{\text{check}J}$ are empty and relations N, C and R are interpreted as in $I_{\mathbf{A}}$.

It is easy to see that J is in the outcome of applying P_1 over I . Now, let \mathcal{S}' be the extension of \mathcal{S} where G^{binary} has an extra attribute, C , and K is an instance over \mathcal{S}' that is just like J except that $G^{\text{binary}K}$ is now the same as G^J (and therefore G^K). By definition we obtain that K is a possible outcome of

applying P_2 over J , and therefore K is in $\text{outcome}_{P_1, P_2}(I)$. Furthermore, one can see that the same instance K is again an outcome of applying P_3 over K , therefore obtaining that $\text{outcome}_{P_1, P_2, P_3}(I)$ is nonempty.

(\Leftarrow) Assume now that there is an instance $L \in \text{outcome}_{P_1, P_2, P_3}(I)$. Then by definition there are instances J and K such that J is in $\text{outcome}_{P_1}(I)$, K is in $\text{outcome}_{P_2}(J)$ and L is in $\text{outcome}_{P_3}(K)$.

Let J^* be the restriction of J over the schema \mathcal{S} . From a simple inspection of P_1 we have that J^* satisfies as well the dependencies in P_1 , so that J^* is in $\text{outcome}_{P_1}(I)$.

Let now \mathcal{S}' be the extension of \mathcal{S} that assigns also attribute C to G^{binary} . Now, since K is an outcome of P_2 over J and P_2 has no scope, if we define K^* as the restriction of K over \mathcal{S}' , then clearly K^* must be in the outcome of applying P_2 over J^* . Note that, by definition of P_3 (since its scope is empty), the restriction of L up to the schema of K must be the same instance as K , and therefore the restriction L^* of L to \mathcal{S}' must be the same instance than K^* . Furthermore, since L (and thus L^*) satisfies the constraints in P_3 , and the constraints only mention relations and atoms in \mathcal{S}' , we have that K^* must be an outcome of applying P_3 over (K^*, \mathcal{S}') .

We now claim that K^* satisfy all tgds (1)-(11) in the proof of Proposition 2. Tgds (1-3) and (6) are immediate from the scopes of procedures, and the satisfaction for all the remaining ones is shown in the same way. For example, to see that K^* satisfies $E(x, y) \rightarrow E(y, x)$, note that J^* already satisfies $E(x, y) \rightarrow E^d(y, x)$. From the fact that the interpretations of E^d and E are the same over J^* and K^* and that K^* satisfies $E^d(x, y) \rightarrow E(x, y)$ we obtain the desired result.

Finally, since K^* satisfies $F(x) \rightarrow F^{\text{check}}(x)$, and the interpretation of F^{check} over all of I , J^* and K^* must be empty, we have that the interpretation of F over K^* is empty as well. Given that K^* satisfies all dependencies in Σ , it must be the case that the left hand side of the tgd (11) is not true K^* , for any possible assignment. By using the same argument as in the proof of Proposition 2 we obtain that $\mathbf{A} = (A, g)$ is embeddable in a finite semigroup.

B.5 Proof of Theorem 1

This theorem is an immediate corollary of Proposition 7, together with an inspection on the complexity of computing the over-approximation. We provide all details in the proof of the next proposition (Proposition 7).

B.6 Proof of Proposition 7

For the proof we assume that all procedures does not use preconditions. We can treat them by first doing an initial check on compatibility that only complicates the proof.

We also specify an alternative set of representatives for conditional instances (which is actually the usual one). The set $\text{rep}(G)$ of representatives of a conditional instance G is simply $\text{rep}(G) = \{I \mid \text{there is a substitution } \nu \text{ such that}$

$\nu(T) \subseteq I\}$. That is, $\hat{rep}(G)$ only specifies instances over the same schema as G . The following lemma allows us to work with this representation instead; it is immediate from the definition of safe scope procedures.

Lemma 2. *If G is a conditional instance, then (1) $\hat{rep}(G) \subseteq rep(G)$, and (2) an instance J is minimal for $rep(G)$ if and only if it is minimal for $\hat{rep}(G)$.*

Moreover, from the fact that procedures with safe scope are acyclic, we can state Theorem 5.1 in [2] in the following terms:

Lemma 3 ([2]). *Given a set Σ of tgds and a positive conditional instance G , one can construct, in polynomial time, a positive conditional instance G' such that (1) $\hat{rep}(G') \subseteq \hat{rep}(G)$ and (2) all minimal models of $\hat{rep}(G')$ satisfy Σ .*

Moreover, by slightly adapting the proof of Proposition 4.6 in [2], we can see that the conditional instance constructed above has even better properties. In order to prove this theorem all that one needs to do is to adapt the notion of solutions for data exchange into a scenario where the target instance may already have some tuples (which will not fire any dependencies because of the safeness of procedures).

Lemma 4 ([2]). *Let $P = (Scope, \mathcal{C}_{in}, \mathcal{C}_{out}, \mathcal{Q}_{safe})$ be a procedure with safe scope, and let G be a positive conditional instance. Then one can construct a conditional instance G' such that, for every minimal instance I of $\hat{rep}(G)$, the set $\hat{rep}(G')$ contains all minimal instances in $outcome_P(I)$, and for every minimal instance J in $\hat{rep}(G')$ there is a minimal instance I of $rep(G)$ such that J is minimal for $outcome_P(I)$.*

Finally, we can show the key result for this proof.

Lemma 5. *Let \mathcal{I} be a set of instances, and G a conditional table that is minimal for \mathcal{I} , and $P = (Scope, \mathcal{C}_{in}, \mathcal{C}_{out}, \mathcal{Q}_{safe})$ a procedure with safe scope. Then either $outcome_P(\mathcal{I}) = \emptyset$ or one can construct, in polynomial time, a conditional instance G' such that*

- i) $outcome_P(\mathcal{I}) \subseteq rep(G')$; and
- ii) *If J is a minimal instance in $rep(G')$, then J is also minimal in $outcome_P(\mathcal{I})$.*

Proof. Using the chase procedure mentioned in Lemma 4, we see that the conditional table G' produced in this lemma satisfies the conditions of this Lemma, for $\hat{rep}(G)$.

For i), let J be an instance in $outcome_P(\mathcal{I})$. Then there is an instance I in \mathcal{I} such that $J \in outcome_P(I)$. Let I^* be a minimal instance in \mathcal{I} such that I extends I^* . By our assumption we know that I^* belongs to $rep(G)$, and since I^* is minimal it must be the case that I^* belongs (and is minimal) for $\hat{rep}(G)$. Therefore, by Lemma 4 we have that $\hat{rep}(G')$ contains all minimal instances for $outcome_P(I^*)$. But now notice that for every assignment τ and tgd λ such that (I^*, τ) satisfies λ , we have that (I, τ) satisfy λ as well. This means that every instance in the set $outcome_P(I)$ must extend a minimal instance in $outcome_P(I^*)$

(if not, then a tgd would not be satisfied due to some assignment that would not be possible to extend). Since every minimal instance in $\text{outcome}_P(I^*)$ is in $\hat{\text{rep}}(G')$, then by the semantics of conditional tables it must be the case that J belongs to $\hat{\text{rep}}(G')$ as well, and therefore to $\text{rep}(G')$.

Item [ii)] follows from the fact that any minimal instance in $\text{rep}(G')$ must also be minimal for $\hat{\text{rep}}(G')$ and a direct application of Lemma 4.

The next Lemma constructs the desired outcomes for alter schema procedures.

Lemma 6. *Let \mathcal{I} be a set of instances, and G a conditional table that is minimal for \mathcal{I} , and $P = (\text{Scope}, \mathcal{C}_{in}, \mathcal{C}_{out}, \mathcal{Q}_{safe})$ an alter schema procedure. Then either $\text{outcome}_P(\mathcal{I}) = \emptyset$ or one can construct, in polynomial time, a conditional instance G' such that*

- i) $\text{outcome}_P(\mathcal{I}) \subseteq \text{rep}(G')$; and*
- ii) If J is a minimal instance in $\text{rep}(G')$, then J is also minimal in $\text{outcome}_P(\mathcal{I})$.*

Proof. Assume that $\text{outcome}_P(\mathcal{I}) \neq \emptyset$ (this can be easily checked in polynomial time). Then one can compute the schema \mathcal{S}_{\min} from the proof of Proposition 3. This schema will add some attributes to some relations in the schema of G , and possibly some other relations with other sets of attributes. Let $\text{Schema}(G) = \mathcal{S}$.

We extend G to a conditional table G' over \mathcal{S}_{\min} as follows:

1. For every relation R such that $\mathcal{S}_{\min}(R) \setminus \mathcal{S}(R) = \{A_1, \dots, A_n\}$, with $n \geq 1$, for tuples from G' by adding to each tuple in G a fresh null value in each of the attributes A_1, \dots, A_n .
2. For every relation R such that $\mathcal{S}(R)$ is not defined, but $\mathcal{S}_{\min}(R)$ is defined, set $R^{G'} = \emptyset$

The properties of the lemma now follow from a straightforward check.

The proof of Proposition 7 now follows from successive applications of Lemmas 6 and 5: one just need to compute the appropriate conditional table for each procedure in the sequence P_1, \dots, P_n . That each construction is in polynomial size if the number n of procedures is fixed, or exponential in other case, follows also from these Lemmas, as the size of the conditional table G' , for a procedure P and a conditional table G , is at most polynomial in G and P (and thus we are composing a polynomial number of polynomials, or a fixed number if n is fixed).

Proof of Theorem 1: While in general checking that the set represented by an arbitrary conditional instance may be np-complete, we note that in [2] it was shown that, for Lemma 5, all that is needed is a positive conditional instance, and clearly deciding whether a positive conditional instance represents at least one solution is in polynomial time. Thus, for the proof of the Theorem we just compute the (positive) conditional instance exhibited for Proposition 7 and then do the polynomial check on the size of the final conditional instance.